Exercise Sheet 2: Answers
COMS10017 Algorithms 2023/2024

Reminder: $\log n$ denotes the binary logarithm, i.e., $\log n = \log_2 n$.

Example Question: Runtime Analysis

Question. What is the runtime of the following algorithm in big-O-notation:

```
Algorithm 1
Require: Integer $n \geq 1$
1: $x \leftarrow 0$
2: for $i = 1 \ldots n$ do
3:     for $j = i \ldots n$ do
4:         $x \leftarrow x + i \cdot j$
5:     end for
6: end for
7: return $x$
```

Solution. We need to sum up the runtimes of all the instructions of Algorithm 1. We account a runtime of $O(1)$ for each of the instructions in Lines 1,4,7, however, the two nested loops make Line 4 being executed multiple times. The runtime of the two nested loops, which dominates the overall runtime of the algorithm, can be computed as follows:

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i}^{n} O(1) &= O\left(\sum_{i=1}^{n}\sum_{j=i}^{n} 1\right) = O\left(\sum_{i=1}^{n} n-i+1\right) = O\left(\sum_{i=1}^{n}(n+1) - \sum_{i=1}^{n} i\right) \\ &= O\left(n(n+1) - \frac{n(n+1)}{2}\right) = O\left(\frac{n(n+1)}{2}\right) = O(\frac{1}{2}n^2 + \frac{1}{2}n) = O(n^2) \ .\end{aligned}$$

The runtime of Algorithm 1 is therefore $O(n^2)$.

Remark: In the previous calculation, we used the simplification $\sum_{j=i}^{n} 1 = n - i + 1$. Observe that j takes on the values $\{i, i+1, \ldots, n\}$, and, for each value, we have a contribution of 1 to the overall sum. Since $|\{i, i+1, \ldots, n\}| = n - i + 1$, i.e., j takes on $n - i + 1$ different values, we obtain the result. We also used the identity $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, which is an important identity that you should remember. In the last step, we used a lemma discussed in the lecture that states that a polynomial in n with constant maximum degree k is in $O(n^k)$. ✓

1 Θ and Ω

1. Prove that the following two statements are equivalent:

(a) $f \in \Theta(g)$.

(b) $f \in O(g)$ and $g \in O(f)$.

Solution. In order to prove that two statements are equivalent, we assume first that the first statement holds and we deduce that the second statement then holds as well. Then we assume that the second statement holds and we deduce that the first statement then holds as well.

Let's first assume that $f \in \Theta(g)$. This means that there are constants c_1, c_2, n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$, for every $n \geq n_0$.

To show that $f \in O(g)$, we need to show that there are constants c, n_0' such that $f(n) \leq c g(n)$, for every $n \geq n_0'$. This follows immediately by choosing $c = c_2$ and $n_0' = n_0$ as above.

To show that $g \in O(f)$, we need to show that there are constants c, n_0' such that $g(n) \leq c f(n)$, for every $n \geq n_0'$. This follows immediately by choosing $c = \frac{1}{c_1}$ and $n \geq n_0'$.

Next, we assume that $f \in O(g)$ and $g \in O(f)$. This implies that there are constants c_1, n_1 such that $f(n) \leq c_1 g(n)$, for every $n \geq n_1$, and constants c_2, n_2 such that $g(n) \leq c_2 f(n)$, for every $n \geq n_2$. We need to show that there are constants d_1, d_2, n_0 such that $d_1 g(n) \leq f(n) \leq d_2 g(n)$, for every $n \geq n_0$. We can chose $d_2 = c_1$, $d_1 = \frac{1}{c_2}$, and $n_0 \geq \max\{n_1, n_2\}$. ✓

2. Prove that the following two statements are equivalent:

(a) $f \in \Omega(g)$.

(b) $g \in O(f)$.

Solution. Let's first assume that $f \in \Omega(g)$. This means that there are constants c_1, n_1 such that $c_1 g(n) \leq f(n)$, for every $n \geq n_1$. We need to show that there are constants c_2, n_2 such that $g(n) \leq c_2 f(n)$, for every $n \geq n_2$. We can pick $c_2 = \frac{1}{c_1}$ and $n_2 = n_1$.

The reverse direction, i.e., assuming that $g \in O(f)$ and deducing that $f \in \Omega(g)$ is very similar. ✓

3. Let $c > 1$ be a constant. Prove or disprove the following statements:

(a) $\log_c n \in \Theta(\log n)$.

Solution. Recall the definition of Θ: A function $f(n) \in \Theta(g(n))$ if there are constants c_1, c_2, n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for every $n \geq n_0$. Hence, we need to find constants c_1, c_2, n_0 such that

$$c_1 \log n \leq \log_c n \leq c_2 \log n \ ,$$

for every $n \geq n_0$. Observe that $\log_c n = \frac{\log n}{\log c}$. We can hence chose $c_1 = c_2 = \frac{1}{\log c}$ and $n_0 = 1$, since $c_1 \cdot \log n = c_2 \cdot \log n = \log_c n$. This clearly holds for every $n \geq 1$. ✓

(b) $\log(n^c) \in \Theta(\log n)$.

Solution. Again, we need to find constants c_1, c_2, n_0 such that

$$c_1 \log n \leq \log(n^c) \leq c_2 \log n \ ,$$

for every $n \geq n_0$. Observe that $\log(n^c) = c \log n$. We can hence chose $c_1 = c_2 = c$ and $n_0 = 1$. ✓

4. Let $c > 2$ be a constant. Prove or disprove the following statement:

$$2^n \in \Theta(c^n) \ .$$

Solution. This statement is wrong. We will show that $c^n \notin O(2^n)$. This disproves this statement since if $f \in \Theta(g)$ then $g \in O(f)$ as well.

For the sake of a contradiction, suppose that $c^n \in O(2^n)$. Then there are constants d, n_0 such that

$$c^n \leq d \cdot 2^n \ ,$$

for every $n \geq n_0$. Taking logarithms on both sides, we obtain the equivalent inequality:

$$\begin{aligned} n \log(c) &\leq \log(d2^n) = \log(d) + n \\ n &\leq \frac{\log(d)}{\log(c) - 1} \ . \end{aligned}$$

Observe that we only obtain the last inequality since $c > 2$ (since $c > 2$ we also have $\log c > 1$ and $\log(c) - 1 > 0$). This inequality hence does not hold for every $n > \frac{\log(d)}{\log(c)-1}$. This is a contradiction to the assumption that it holds for every $n \geq n_0$. ✓

2 O-notation

1. Consider the following functions:

$$f_1 = 2^{\sqrt{n}}, f_2 = \log^2(20n), f_3 = n!, f_4 = \frac{1}{2}n^2/\log(n), f_5 = 4\log^2(n), f_6 = 2^{\sqrt{\log n}} \ .$$

Relabel the functions such that $f_i \in O(f_{i+1})$ (no need to give any proofs here).

Solution.

$$O(\log^2(20n)) \subseteq O(4\log^2(n)) \subseteq O(2^{\sqrt{\log n}}) \subseteq O(\frac{1}{2}n^2/\log(n)) \subseteq O(2^{\sqrt{n}}) \subseteq O(n!)$$

Observe that $\log^2(20n) = \Theta(4\log^2(n))$. We could therefore also swap the positions of the first two functions. ✓

2. Give functions f, g such that $f(n) \in O(g(n))$ and $2^{f(n)} \notin O(2^{g(n)})$.

Solution. Consider for example $f(n) = \log n$ and $g(n) = \frac{1}{2}\log n$. Then clearly $f(n) = O(g(n))$, but $2^{f(n)} \notin 2^{g(n)}$, since $2^{f(n)} = 2^{\log n} = n$ and $2^{g(n)} = 2^{\frac{1}{2}\log n} = n^{\frac{1}{2}} = \sqrt{n}$, and $n \notin O(\sqrt{n})$. ✓

3 Runtime Analysis

Algorithm 2

```
Require: Int $n \geq 1$
x ← 0
for $i = 1 \dots n$ do
    for $j = 1 \dots n$ do
        for $k = 1 \dots n$ do
            $x \leftarrow x + i \cdot j$
        end for
    end for
end for
return $x$
```

Algorithm 3

```
Require: Int $n \geq 1$
$x \leftarrow 0$
$i \leftarrow 1$
while $i \leq n$ do
    for $j = 1 \dots n$ do
        $x \leftarrow x + i \cdot j$
    end for
    $i \leftarrow 2 \cdot i$
end while
return $x$
```

Algorithm 4

```
Require: Int $n \geq 1$
$x \leftarrow 0$
$i \leftarrow 1$
while $i \leq n$ do
    for $j = 1 \dots i$ do
        $x \leftarrow x + i \cdot j$
    end for
    $i \leftarrow 2 \cdot i$
end while
return $x$
```

Determine the runtimes of Algorithms 2, 3, and 4 using big-O-notation.

Solution.

1. Algorithm 1 runs in time $\Theta(n^3)$ (three nested loops, each going from 1 to n).

2. Observe that the inner loop in Algorithm 3 always requires $\Theta(n)$ time in total. It remains to determine how often the outer loop is executed. To this end, for $j \geq 1$, let i_j be the value of i at the beginning of iteration j. Then, $i_1 = 1$, and $i_j = 2 \cdot i_{j-1} = 4 \cdot i_{j-2} = \dots = 2^{j-1}$. Let k be the last iteration of the loop. Then, $2^{k-1} \leq n$ and $2^k > n$. We have $2^k > n \Rightarrow k > \log n$, and similarly we get $k - 1 \leq \log n$, which implies $k \leq \log(n) + 1$. We thus have the conditions: $\log n < k < \log(n) + 1$, and since k is an integer, we obtain $k = \lfloor \log(n) + 1 \rfloor$. Hence, the outer loop is executed $\lfloor \log(n) + 1 \rfloor$ times. The total runtime is therefore $\Theta(n \log n)$.

3. Observe that in Algorithm 4 the inner loop runs in time $\Theta(i)$. As demonstrated in the previous exercise, the outer loop is executed $\lfloor \log(n) + 1 \rfloor$ times and the variable i takes on values $1, 2, 4, 8, \dots, 2^{\lfloor \log(n)+1 \rfloor - 1}$. We can thus bound the runtime as follows:

$$\Theta\left(\sum_{j=1}^{\lfloor \log(n)+1 \rfloor - 1} 2^j \right) = \Theta(2^{\lfloor \log(n)+1 \rfloor}) = \Theta(n).$$

✓

4 Optional and Difficult Questions

Exercises in this section are intentionally more difficult and are there to challenge yourself.

4.1 Peak Finding in 2D (hard!)

Let A be an n-by-n matrix of integers, for some integer n. We say that $A_{i,j}$ is a *peak* if the adjacent elements $A_{i-1,j}, A_{i+1,j}, A_{i,j-1}, A_{i,j+1}$ are not larger than $A_{i,j}$. The objective is to find a peak in A. Similar to the peak finding problem discussed in the lecture, reporting any peak is fine, in particular, it is not required that we find the maximum in A or that we report all the peaks in A.

Consider the following baseline algorithm: We scan the entire matrix and check whether every element $A_{i,j}$, for $i, j \in \{0, 1, 2, \ldots, n-1\}$, is a peak. This strategy requires a runtime of $O(n^2)$. Is there a faster algorithm?

Please send your ideas to `christian.konrad@bristol.ac.uk`. I am keen to hear if you found a solution!